Finite State Transducers Approximating Hidden Markov Models

André Kempe
Rank Xerox Research Centre – Grenoble Laboratory
6, chemin de Maupertuis – 38240 Meylan – France
andre.kempe@grenoble.rxrc.xerox.com
http://www.rxrc.xerox.com/research/mltt

Abstract

This paper describes the conversion of a Hidden Markov Model into a sequential transducer that closely approximates the behavior of the stochastic model. This transformation is especially advantageous for part-of-speech tagging because the resulting transducer can be composed with other transducers that encode correction rules for the most frequent tagging errors. The speed of tagging is also improved. The described methods have been implemented and successfully tested on six languages.

1 Introduction

Finite-state automata have been successfully applied in many areas of computational linguistics.

This paper describes two algorithms[1] which approximate a *Hidden Markov Model* (HMM) used for part-of-speech tagging by a *finite-state transducer* (FST). These algorithms may be useful beyond the current description on any kind of analysis of written or spoken language based on both finite-state technology and HMMs, such as corpus analysis, speech recognition, etc. Both algorithms have been fully implemented.

An HMM used for tagging encodes, like a transducer, a relation between two languages. One language contains sequences of ambiguity classes obtained by looking up in a lexicon all words of a sentence. The other language contains sequences of tags obtained by statistically disambiguating the class sequences. From the outside, an HMM tagger behaves like a sequential transducer that deterministically maps every class sequence to a tag sequence, e.g.:

$$\frac{\texttt{[DET,PRO]}\ \ \texttt{[ADJ,NOUN]}\ \ \texttt{[ADJ,NOUN]}\ \ \ldots\ldots\ \ \texttt{[END]}}{\texttt{DET}\qquad\quad \texttt{ADJ}\qquad\quad \texttt{NOUN}\qquad \ldots\ldots\ \ \texttt{END}} \tag{1}$$

The aim of the conversion is not to generate FSTs that behave in the same way, or in as similar a way as possible like HMMs, but rather FSTs that perform tagging in as accurate a way as possible. The motivation to derive these FSTs from HMMs is that HMMs can be trained and converted with little manual effort.

The tagging speed when using transducers is up to five times higher than when using the underlying HMMs. The main advantage of transforming an HMM is that the resulting transducer can be handled by finite state calculus. Among others, it can be composed with transducers that encode:

- correction rules for the most frequent tagging errors which are automatically generated (Brill, 1992; Roche and Schabes, 1995) or manually written (Chanod and Tapanainen, 1995), in order to significantly improve tagging accuracy[2]. These rules may include long-distance dependencies not handled by HMM taggers, and can conveniently be expressed by the replace operator (Kaplan and Kay, 1994; Karttunen, 1995; Kempe and Karttunen, 1996).
- further steps of text analysis, e.g. light parsing or extraction of noun phrases or other phrases (Aït-Mokhtar and Chanod, 1997).

These compositions enable complex text analysis to be performed by a single transducer.

An HMM transducer builds on the data (probability matrices) of the underlying HMM. The accuracy

[1] There is a different (unpublished) algorithm by Julian M. Kupiec and John T. Maxwell (p.c.).

[2] Automatically derived rules require less work than manually written ones but are unlikely to yield better results because they would consider relatively limited context and simple relations only.

of this data has an impact on the tagging accuracy of both the HMM itself and the derived transducer. The training of the HMM can be done on either a tagged or untagged corpus, and is not a topic of this paper since it is exhaustively described in the literature (Bahl and Mercer, 1976; Church, 1988).

An HMM can be identically represented by a weighted FST in a straightforward way. We are, however, interested in non-weighted transducers.

2 n-Type Approximation

This section presents a method that approximates a (1st order) HMM by a transducer, called *n-type* approximation[3].

Like in an HMM, we take into account initial probabilities π, transition probabilities a and class (i.e. observation symbol) probabilities b. We do, however, not estimate probabilities over paths. The tag of the first word is selected based on its initial and class probability. The next tag is selected on its transition probability given the first tag, and its class probability, etc. Unlike in an HMM, once a decision on a tag has been made, it influences the following decisions but is itself irreversible.

A transducer encoding this behaviour can be generated as sketched in figure 1. In this example we have a set of three classes, c_1 with the two tags t_{11} and t_{12}, c_2 with the three tags t_{21}, t_{22} and t_{23}, and c_3 with one tag t_{31}. Different classes may contain the same tag, e.g. t_{12} and t_{23} may refer to the same tag.

For every possible pair of a class and a tag (e.g. $c_1 : t_{12}$ or `[ADJ,NOUN]:NOUN`) a state is created and labelled with this same pair (fig. 1). An initial state which does not correspond with any pair, is also created. All states are final, marked by double circles.

For every state, as many outgoing arcs are created as there are classes (three in fig. 1). Each such arc for a particular class points to the most probable pair of this same class. If the arc comes from the initial state, the most probable pair of a class and a tag (destination state) is estimated by:

$$\arg\max_k p_1(c_i, t_{ik}) = \pi(t_{ik})\; b(c_i|t_{ik}) \qquad (2)$$

If the arc comes from a state other than the initial state, the most probable pair is estimated by:

$$\arg\max_k p_2(c_i, t_{ik}) = a(t_{ik}|t_{previous})\; b(c_i|t_{ik}) \qquad (3)$$

In the example (fig. 1) $c_1 : t_{12}$ is the most likely pair of class c_1, and $c_2 : t_{23}$ the most likely pair of class c_2 when coming from the initial state, and $c_2 : t_{21}$ the most likely pair of class c_2 when coming from the state of $c_3 : t_{31}$.

Every arc is labelled with the same symbol pair as its destination state, with the class symbol in the upper language and the tag symbol in the lower language. E.g. every arc leading to the state of $c_1 : t_{12}$ is labelled with $\mathtt{c_1 : t_{12}}$.

Finally, all state labels can be deleted since the behaviour described above is encoded in the arc labels and the network structure. The network can be minimized and determinized.

We call the model an *n1-type model*, the resulting FST an *n1-type transducer* and the algorithm leading from the HMM to this transducer, an *n1-type approximation* of a 1st order HMM.

Adapted to a 2nd order HMM, this algorithm would give an *n2-type approximation*. Adapted to a zero order HMM, which means only to use class probabilities b, the algorithm would give an *n0-type approximation*.

n-Type transducers have deterministic states only.

3 s-Type Approximation

This section presents a method that approximates an HMM by a transducer, called *s-type* approximation[4].

Tagging a sentence based on a 1st order HMM includes finding the most probable tag sequence T given the class sequence C of the sentence. The joint probability of C and T can be estimated by:

$$p(C,T) = p(c_1....c_n, t_1....t_n) = \pi(t_1)\; b(c_1|t_1) \cdot \prod_{i=2}^{n} a(t_i|t_{i-1})\; b(c_i|t_i) \qquad (4)$$

The decision on a tag of a particular word cannot be made separately from the other tags. Tags can influence each other over a long distance via transition probabilities. Often, however, it is unnecessary to decide on the tags of the whole sentence at once. In the case of a 1st order HMM, unambiguous classes (containing one tag only), plus the sentence beginning and end positions, constitute barriers to the propagation of HMM probabilities. Two tags with one or more barriers inbetween do not influence each other's probability.

[3] Name given by the author.

[4] Name given by the author.

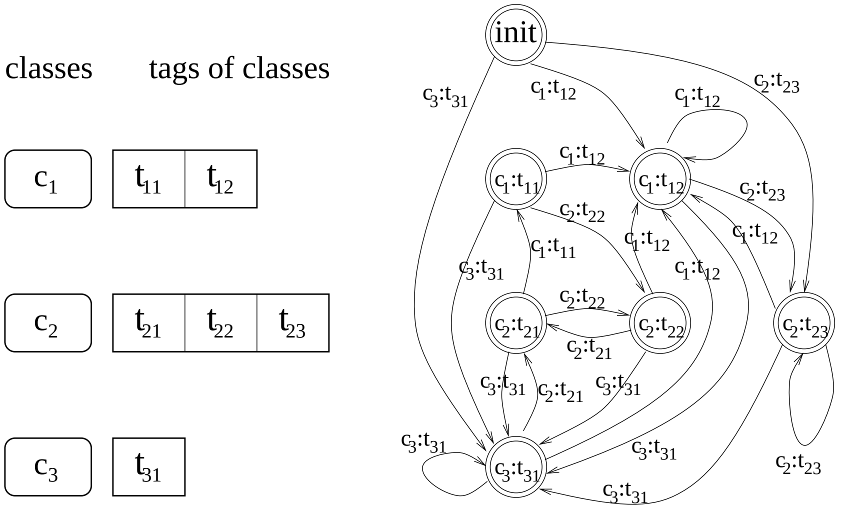

Figure 1: Generation of an n1-type transducer

3.1 s-Type Sentence Model

To tag a sentence, one can split its class sequence at the barriers into subsequences, then tag them separately and concatenate them again. The result is equivalent to the one obtained by tagging the sentence as a whole.

We distinguish between initial and middle subsequences. The final subsequence of a sentence is equivalent to a middle one, if we assume that the sentence end symbol (. or ! or ?) always corresponds to an unambiguous class c_u. This allows us to ignore the meaning of the sentence end position as an HMM barrier because this role is taken by the unambiguous class c_u at the sentence end.

An initial subsequence C_i starts with the sentence initial position, has any number (incl. zero) of ambiguous classes c_a and ends with the first unambiguous class c_u of the sentence. It can be described by the regular expression[5]:

$$C_i = c_a{*}\; c_u \tag{5}$$

The joint probability of an initial class subsequence C_i of length r, together with an initial tag subsequence T_i, can be estimated by:

$$p(C_i, T_i) = \pi(t_1)\; b(c_1|t_1) \cdot \prod_{j=2}^{r} a(t_j|t_{j-1})\; b(c_j|t_j) \tag{6}$$

A middle subsequence C_m starts immediately after an unambiguous class c_u, has any number (incl. zero) of ambiguous classes c_a and ends with the following unambiguous class c_u:

$$C_m = c_a{*}\; c_u \tag{7}$$

For correct probability estimation we have to include the immediately preceding unambiguous class c_u, actually belonging to the preceding subsequence C_i or C_m. We thereby obtain an extended middle subsequence[5]:

$$C_m^e = c_u^e\; c_a{*}\; c_u \tag{8}$$

The joint probability of an extended middle class subsequence C_m^e of length s, together with a tag subsequence T_m^e, can be estimated by:

$$p(C_m^e, T_m^e) = b(c_1|t_1) \cdot \prod_{j=2}^{s} a(t_j|t_{j-1})\; b(c_j|t_j) \tag{9}$$

3.2 Construction of an s-Type Transducer

To build an s-type transducer, a large number of initial class subsequences C_i and extended middle class subsequences C_m^e are generated in one of the following two ways:

(a) Extraction from a corpus

Based on a lexicon and a guesser, we annotate an untagged training corpus with class labels. From every sentence, we extract the initial class subsequence C_i that ends with the first unambiguous class c_u (eq. 5), and all extended middle subsequences C_m^e ranging from any unambiguous class c_u (in the sentence) to the following unambiguous class (eq. 8).

[5] Regular expression operators used in this section are explained in the annex.

A frequency constraint (threshold) may be imposed on the subsequence selection, so that the only subsequences retained are those that occur at least a certain number of times in the training corpus[6].

(b) Generation of possible subsequences

Based on the set of classes, we generate all possible initial and extended middle class subsequences, C_i and C_m^e (eq. 5, 8) up to a defined length.

Every class subsequence C_i or C_m^e is first disambiguated based on a 1st order HMM, using the Viterbi algorithm (Viterbi, 1967; Rabiner, 1990) for efficiency, and then linked to its most probable tag subsequence T_i or T_m^e by means of the cross product operation[5]:

$$S_i = C_i \text{ .x. } T_i = c_1\!:\!t_1 \; c_2\!:\!t_2 \; ...... \; c_n\!:\!t_n \qquad (10)$$

$$S_m^e = C_m^e \text{ .x. } T_m^e = c_1^e\!:\!t_1^e \; c_2\!:\!t_2 \; ...... \; c_n\!:\!t_n \qquad (11)$$

In all extended middle subsequences S_m^e, e.g.:

$$S_m^e = \frac{C_m^e}{T_m^e} = \qquad (12)$$

$$\frac{\texttt{[DET]} \;\; \texttt{[ADJ,NOUN]} \;\; \texttt{[ADJ,NOUN]} \;\; \texttt{[NOUN]}}{\texttt{DET} \qquad \texttt{ADJ} \qquad\quad \texttt{ADJ} \qquad\quad \texttt{NOUN}}$$

the first class symbol on the upper side and the first tag symbol on the lower side, will be marked as an extension that does not really belong to the middle sequence but which is necessary to disambiguate it correctly. Example (12) becomes:

$$S_m^0 = \frac{C_m^0}{T_m^0} = \qquad (13)$$

$$\frac{\texttt{0.[DET]} \;\; \texttt{[ADJ,NOUN]} \;\; \texttt{[ADJ,NOUN]} \;\; \texttt{[NOUN]}}{\texttt{0.DET} \qquad \texttt{ADJ} \qquad\quad \texttt{ADJ} \qquad\quad \texttt{NOUN}}$$

We then build the union ${}^{\cup}S_i$ of all initial subsequences S_i and the union ${}^{\cup}S_m^e$ of all extended middle subsequences S_m^e, and formulate a preliminary sentence model:

$${}^{\cup}S^0 \;=\; {}^{\cup}S_i \;\; {}^{\cup}S_m^0{*} \qquad (14)$$

in which all middle subsequences S_m^0 are still marked and extended in the sense that all occurrences of all unambiguous classes are mentioned twice: Once unmarked as c_u at the end of every sequence C_i or C_m^0, and the second time marked as c_u^0 at the beginning of every following sequence C_m^0. The upper side of the sentence model ${}^{\cup}S^0$ describes the complete (but extended) class sequences of possible sentences, and the lower side of ${}^{\cup}S^0$ describes the corresponding (extended) tag sequences.

To ensure a correct concatenation of initial and middle subsequences, we formulate a concatenation constraint for the classes:

$$R_c = \bigcap_j \; [\,{\sim}\$[\ \backslash c_u \; c_u^0 \ ]\ ]_j \qquad (15)$$

stating that every middle subsequence must begin with the same marked unambiguous class c_u^0 (e.g. `0.[DET]`) which occurs unmarked as c_u (e.g. `[DET]`) at the end of the preceding subsequence since both symbols refer to the same occurrence of this unambiguous class.

Having ensured correct concatenation, we delete all marked classes on the upper side of the relation by means of

$$D_c = [\,] \;{<}{-}\; \left[\bigcup_j [c_u^0]_j \right] \qquad (16)$$

and all marked tags on the lower side by means of

$$D_t = \left[\bigcup_j [t^0]_j \right] \;{-}{>}\; [\,] \qquad (17)$$

By composing the above relations with the preliminary sentence model, we obtain the final sentence model[5]:

$$S = D_c \text{ .o. } R_c \text{ .o. } {}^{\cup}S^0 \text{ .o. } D_t \qquad (18)$$

We call the model an *s-type model*, the corresponding FST an *s-type transducer*, and the whole algorithm leading from the HMM to the transducer, an *s-type approximation* of an HMM.

The s-type transducer tags any corpus which contains only known subsequences, in exactly the same way, i.e. with the same errors, as the corresponding HMM tagger does. However, since an s-type transducer is incomplete, it cannot tag sentences with one or more class subsequences not contained in the union of the initial or middle subsequences.

3.3 Completion of an s-Type Transducer

An incomplete s-type transducer S can be completed with subsequences from an auxiliary, complete n-type transducer N as follows:

First, we extract the union of initial and the union of extended middle subsequences, ${}^{\cup}_{s}S_i$ and ${}^{\cup}_{s}S_m^e$ from

[6] The frequency constraint may prevent the encoding of rare subsequences which would encrease the size of the transducer without contributing much to the tagging accuracy.

the primary s-type transducer S, and the unions ${}^{\cup}_{n}S_i$ and ${}^{\cup}_{n}S^e_m$ from the auxiliary n-type transducer N. To extract the union ${}^{\cup}S_i$ of initial subsequences we use the following filter:

$$F_{S_i} = [\ \backslash\langle c_u, t\rangle\ ]*\ \ \langle c_u, t\rangle\ \ [\ ?:[\ ]\ ]* \qquad (19)$$

where $\langle c_u, t\rangle$ is the 1-level format[7] of the symbol pair $c_u\!:\!t$. The extraction takes place by

$${}^{\cup}S_i = [\ N.1L\ .\mathrm{o}.\ F_{S_i}\ ].l.2L \qquad (20)$$

where the transducer N is first converted into 1-level format[7], then composed with the filter F_{S_i} (eq. 19). We extract the lower side of this composition, where every sequence of $N.1L$ remains unchanged from the beginning up to the first occurrence of an unambiguous class c_u. Every following symbol is mapped to the empty string by means of $[?:[\ ]]*$ (eq. 19). Finally, the extracted lower side is again converted into 2-level format[7].

The extraction of the union ${}^{\cup}S^e_m$ of extended middle subsequences is performed in a similar way.

We then make the joint unions of initial and extended middle subsequences[5]:

$${}^{\cup}S_i = {}^{\cup}_{s}S_i \mid [\ [\ {}^{\cup}_{n}S_i.u - {}^{\cup}_{s}S_i.u\ ]\ .\mathrm{o}.\ {}^{\cup}_{n}S_i\ ] \qquad (21)$$

$${}^{\cup}S^e_m = {}^{\cup}_{s}S^e_m \mid [\ [\ {}^{\cup}_{n}S^e_m.u - {}^{\cup}_{s}S^e_m.u\ ]\ .\mathrm{o}.\ {}^{\cup}_{n}S^e_m\ ] \qquad (22)$$

In both cases (eq. 21 and 22) we union all subsequences from the principal model S, with all those subsequences from the auxiliary model N that are not in S.

Finally, we generate the completed *s+n-type* transducer from the joint unions of subsequences ${}^{\cup}S_i$ and ${}^{\cup}S^e_m$, as decribed above (eq. 14-18).

A transducer completed in this way, disambiguates all subsequences known to the principal incomplete s-type model, exactly as the underlying HMM does, and all other subsequences as the auxiliary n-type model does.

4 An Implemented Finite-State Tagger

The implemented tagger requires three transducers which represent a lexicon, a guesser and any above mentioned approximation of an HMM.

All three transducers are sequential, i.e. deterministic on the input side.

Both the lexicon and guesser unambiguously map a surface form of any word that they accept to the corresponding class of tags (fig. 2, col. 1 and 2): First, the word is looked for in the lexicon. If this fails, it is looked for in the guesser. If this equally fails, it gets the label `[UNKNOWN]` which associates the word with the tag class of unknown words. Tag probabilities in this class are approximated by tags of words that appear only once in the training corpus.

As soon as an input token gets labelled with the tag class of sentence end symbols (fig. 2: `[SENT]`), the tagger stops reading words from the input. At this point, the tagger has read and stored the words of a whole sentence (fig. 2, col. 1) and generated the corresponding sequence of classes (fig. 2, col. 2).

The class sequence is now deterministically mapped to a tag sequence (fig. 2, col. 3) by means of the HMM transducer. The tagger outputs the stored word and tag sequence of the sentence, and continues in the same way with the remaining sentences of the corpus.

The	[AT]	AT
share	[NN,VB]	NN
of	[IN]	IN
...	...	...
tripled	[VBD,VBN]	VBD
within	[IN,RB]	IN
that	[CS,DT,WPS]	DT
span	[NN,VB,VBD]	VBD
of	[IN]	IN
time	[NN,VB]	NN
.	[SENT]	SENT

Figure 2: Tagging a sentence

5 Experiments and Results

This section compares different n-type and s-type transducers with each other and with the underlying HMM.

The FSTs perform tagging faster than the HMMs.

Since all transducers are approximations of HMMs, they give a lower tagging accuracy than the corresponding HMMs. However, improvement in accuracy can be expected since these transducers can be composed with transducers encoding correction rules for frequent errors (sec. 1).

Table 1 compares different transducers on an English test case.

The s+n1-type transducer containing all possible subsequences up to a length of three classes is the most accurate (table 1, last line, s+n1-FST ($\leq$ 3): 95.95 %) but also the largest one. A similar rate of accuracy at a much lower size can be achieved with

[7] 1-Level and 2-level format are explained in the annex.

	accuracy	tagging speed	transducer size		creation
	in %	in words/sec	# states	# arcs	time
HMM	96.77	4 590	——	——	——
n0-FST	83.53	**20 582**	1	297	16 sec
n1-FST	94.19	17 244	71	21 087	17 sec
s+n1-FST (20K, F1)	94.74	13 575	927	203 853	3 min
s+n1-FST (50K, F1)	94.92	12 760	2 675	564 887	10 min
s+n1-FST (100K, F1)	95.05	12 038	4 709	976 785	23 min
s+n1-FST (100K, F2)	94.76	14 178	476	107 728	2 min
s+n1-FST (100K, F4)	94.60	14 178	211	52 624	76 sec
s+n1-FST (100K, F8)	94.49	13 870	154	41 598	62 sec
s+n1-FST (1M, F2)	95.67	11 393	2 049	418 536	7 min
s+n1-FST (1M, F4)	95.36	11 193	799	167 952	4 min
s+n1-FST (1M, F8)	95.09	13 575	432	96 712	3 min
s+n1-FST ($\leq$ 2)	95.06	8 180	9 796	1 311 962	39 min
s+n1-FST ($\leq$ 3)	**95.95**	4 870	92 463	13 681 113	47 h

Language:	English
Corpora:	19 944 words for HMM training, 19 934 words for test
Tag set:	74 tags 297 classes
Types of FST (Finite-State Transducers) :	
n0, n1	n0-type (with only lexical probabilities) or n1-type (sec. 2)
s+n1 (100K, F2)	s-type (sec. 3), with subsequences of frequency $\geq$ 2, from a training corpus of 100 000 words (sec. 3.2 a), completed with n1-type (sec. 3.3)
s+n1 ($\leq$ 2)	s-type (sec. 3), with all possible subsequences of length $\leq$ 2 classes (sec. 3.2 b), completed with n1-type (sec. 3.3)
Computer:	ultra2, 1 CPU, 512 MBytes physical RAM, 1.4 GBytes virtual RAM

Table 1: Accuracy, speed, size and creation time of some HMM transducers

the s+n1-type, either with all subsequences up to a length of two classes (s+n1-FST ($\leq$ 2): 95.06 %) or with subsequences occurring at least once in a training corpus of 100 000 words (s+n1-FST (100K, F1): 95.05 %).

Increasing the size of the training corpus and the frequency limit, i.e. the number of times that a subsequence must at least occur in the training corpus in order to be selected (sec. 3.2 a), improves the relation between tagging accuracy and the size of the transducer. E.g. the s+n1-type transducer that encodes subsequences from a training corpus of 20 000 words (table 1, s+n1-FST (20K, F1): 94.74 %, 927 states, 203 853 arcs), performs less accurate tagging and is bigger than the transducer that encodes subsequences occurring at least eight times in a corpus of 1 000 000 words (table 1, s+n1-FST (1M, F8): 95.09 %, 432 states, 96 712 arcs).

Most transducers in table 1 are faster then the underlying HMM; the n0-type transducer about five times[8]. There is a large variation in speed between the different transducers due to their structure and size.

Table 2 compares the tagging accuracy of different transducers and the underlying HMM for different languages. In these tests the highest accuracy was always obtained by s-type transducers, either with all subsequences up to a length of two classes[9] or with subsequences occurring at least once in a corpus of 100 000 words.

6 Conclusion and Future Research

The two methods described in this paper allow the approximation of an HMM used for part-of-speech tagging, by a finite-state transducer. Both methods have been fully implemented.

The tagging speed of the transducers is up to five times higher than that of the underlying HMM.

The main advantage of transforming an HMM

[8] Since n0-type and n1-type transducers have deterministic states only, a particular fast matching algorithm can be used for them.

[9] A maximal length of three classes is not considered here because of the high increase in size and a low increase in accuracy.

	accuracy in %					
	English	Dutch	French	German	Portug.	Spanish
HMM	96.77	94.76	98.65	97.62	97.12	97.60
n0-FST	83.53	81.99	91.13	82.97	91.03	93.65
n1-FST	94.19	91.58	98.18	94.49	96.19	96.46
s+n1-FST (20K, F1)	94.74	92.17	98.35	95.23	96.33	96.71
s+n1-FST (50K, F1)	94.92	92.24	**98.37**	95.57	96.49	96.76
s+n1-FST (100K, F1)	**95.05**	**92.36**	**98.37**	95.81	**96.56**	**96.87**
s+n1-FST (100K, F2)	94.76	92.17	98.34	95.51	96.42	96.74
s+n1-FST (100K, F4)	94.60	92.02	98.30	95.29	96.27	96.64
s+n1-FST (100K, F8)	94.49	91.84	98.32	95.02	96.23	96.54
s+n1-FST (≤ 2)	**95.06**	92.25	**98.37**	**95.92**	96.50	**96.90**
HMM train.crp. (#wd)	19 944	26 386	22 622	91 060	20 956	16 221
test corpus (# words)	19 934	10 468	6 368	39 560	15 536	15 443
# tags	74	47	45	66	67	55
# classes	297	230	287	389	303	254
Types of FST (Finite-State Transducers) :	cf. table 1					

Table 2: Accuracy of some HMM transducers for different languages

is that the resulting FST can be handled by finite state calculus[10] and thus be directly composed with other transducers which encode tag correction rules and/or perform further steps of text analysis.

Future research will mainly focus on this possibility and will include composition with, among others:

- Transducers that encode correction rules (possibly including long-distance dependencies) for the most frequent tagging errors, in order to significantly improve tagging accuracy. These rules can be either extracted automatically from a corpus (Brill, 1992) or written manually (Chanod and Tapanainen, 1995).

- Transducers for light parsing, phrase extraction and other analysis (Aït-Mokhtar and Chanod, 1997).

An HMM transducer can be composed with one or more of these transducers in order to perform complex text analysis using only a single transducer.

We also hope to improve the n-type model by using look-ahead to the following tags[11].

Acknowledgements

I wish to thank the anonymous reviewers of my paper for their valuable comments and suggestions.

I am grateful to Lauri Karttunen and Gregory Grefenstette (both RXRC Grenoble) for extensive and frequent discussion during the period of my work, as well as to Julian Kupiec (Xerox PARC) and Mehryar Mohri (AT&T Research) for sending me some interesting ideas before I started.

Many thanks to all my colleagues at RXRC Grenoble who helped me in whatever respect, particularly to Anne Schiller, Marc Dymetman and Jean-Pierre Chanod for discussing parts of the work, and to Irene Maxwell for correcting various versions of the paper.

[10] A large library of finite-state functions is available at Xerox.

[11] Ongoing work has shown that, looking ahead to just one tag is worthless because it makes tagging results highly ambiguous.

References

Aït-Mokhtar, Salah and Chanod, Jean-Pierre (1997). Incremental Finite-State Parsing. In the *Proceedings of the 5th Conference of Applied Natural Language Processing.* ACL, pp. 72-79. Washington, DC, USA.

Bahl, Lalit R. and Mercer, Robert L. (1976). Part of Speech Assignment by a Statistical Decision Algorithm. In *IEEE international Symposium on Information Theory*. pp. 88-89. Ronneby.

Brill, Eric (1992). A Simple Rule-Based Part-of-Speech Tagger. In the *Proceedings of the 3rd conference on Applied Natural Language Processing*, pp. 152-155. Trento, Italy.

Chanod, Jean-Pierre and Tapanainen, Pasi (1995). Tagging French - Comparing a Statistical and a Constraint Based Method. In the *Proceedings of the 7th conference of the EACL*, pp. 149-156. ACL. Dublin, Ireland.

Church, Kenneth W. (1988). A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing.* ACL, pp. 136-143.

Kaplan, Ronald M. and Kay, Martin (1994). Regular Models of Phonological Rule Systems. In *Computational Linguistics*. 20:3, pp. 331-378.

Karttunen, Lauri (1995). The Replace Operator. In the *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, MA, USA. `cmp-lg/9504032`

Kempe, André and Karttunen, Lauri (1996). Parallel Replacement in Finite State Calculus. In the *Proceedings of the 16th International Conference on Computational Linguistics*, pp. 622-627. Copenhagen, Denmark. `cmp-lg/9607007`

Rabiner, Lawrence R. (1990). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Readings in Speech Recognition* (eds. A. Waibel, K.F. Lee). Morgan Kaufmann Publishers, Inc. San Mateo, CA., USA.

Roche, Emmanuel and Schabes, Yves (1995). Deterministic Part-of-Speech Tagging with Finite-State Transducers. In *Computational Linguistics*. Vol. 21, No. 2, pp. 227-253.

Viterbi, A.J. (1967). Error Bounds for Convolutional Codes and an Asymptotical Optimal Decoding Algorithm. In *Proceedings of IEEE*, vol. 61, pp. 268-278.

Annex: **Regular Expression Operators**

Below, `a` and `b` designate symbols, `A` and `B` designate languages, and `R` and `Q` designate relations between two languages. More details on the following operators and pointers to finite-state literature can be found in `http://www.rxrc.xerox.com/research/mltt/fst`

`$A` Contains. Set of strings containing at least one occurrence of a string from `A` as a substring.

`~A` Complement (negation). All strings except those from `A`.

`\a` Term complement. Any symbol other than `a`.

`A*` Kleene star. Zero or more times `A` concatenated with itself.

`A+` Kleene plus. One or more times `A` concatenated with itself.

`a -> b` Replace. Relation where every `a` on the upper side gets mapped to a `b` on the lower side.

`a <- b` Inverse replace. Relation where every `b` on the lower side gets mapped to an `a` on the upper side.

`a:b` Symbol pair with `a` on the upper and `b` on the lower side.

$\langle a, b\rangle$ 1-Level symbol which is the 1-level form (*.1L*) of the symbol pair `a:b`.

`R.u` Upper language of `R`.

`R.l` Lower language of `R`.

`A  B` Concatenation of all strings of `A` with all strings of `B`.

`A | B` Union of `A` and `B`.

`A & B` Intersection of `A` and `B`.

`A - B` Relative complement (minus). All strings of `A` that are not in `B`.

`A .x. B` Cross Product (Cartesian product) of the languages `A` and `B`.

`R .o. Q` Composition of the relations `R` and `Q`.

`R`*.1L* 1-Level form. Makes a language out of the relation `R`. Every symbol pair becomes a simple symbol. (e.g. `a:b` becomes $\langle a, b\rangle$ and `a` which means `a:a` becomes $\langle a, a\rangle$)

`A`*.2L* 2-Level form. Inverse operation to *.1L* (`R`*.1L.2L* = `R`).

`0` *or* `[ ]` Empty string (epsilon).

`?` Any symbol in the known alphabet and its extensions